Aufgabe 2: Twist

Team-ID: 00920

Team-Name: Paddy Jo

Bearbeiter/-innen dieser Aufgabe:
Patrick Müller, Josua Kugler

17. März 2020

Inhaltsverzeichnis

1 Lösungsidee **1**

2 Umsetzung **1**

3 Beispiele **2**
3.1 Beispiel 2
3.2 Beispiel 3
3.3 Beispiel 3
3.4 Beispiel 3
3.5 Beispiel 3

4 Quellcode **3**

1 Lösungsidee

Zunächst wird die Wörterliste nach Länge, Anfangsbuchstabe und Endbuchstabe sortiert. Nun wird der Text „getwistet“. Alle Wörter mit weniger als vier Buchstaben sowie Leerzeichen, Sonderzeichen oder Satzzeichen werden ohne Änderung so stehen gelassen. Dann werden bei den Wörtern mit mehr als drei Buchstaben die mittleren Buchstaben vertauscht und sie werden wieder in den Text eingefügt.

Um den Text zu „enttwisten“ wird nach dem gleichen Schema verfahren. Allerdings funktioniert das „Enttwisten“ einzelner Wörter anders. Zunächst wird überprüft, ob der Anfangsbuchstabe des Wortes ein Großbuchstabe ist. In der Wörterliste wird die Großschreibung am Satzanfang nicht berücksichtigt, daher wird im Wörterdictionary anhand der Länge und des Anfangs- sowie Endbuchstabens (bei großem Anfangsbuchstabe sowohl mit großem als auch mit kleinem Anfangsbuchstaben) auf die Liste mit möglichen Wörtern zugegriffen. Für jedes Wort in dieser Liste wird überprüft, ob die mittleren Buchstaben ein Anagramm zu den mittleren Buchstaben des „getwisteten“ Wortes bilden. Alle Wörter, für die auch dieses Kriterium zutrifft, werden zu einer neuen Liste hinzugefügt. Eine Ausnahme bilden die Wörter mit großem Anfangsbuchstaben. Da ein Wort, das in der Liste steht, nicht zwingend einen großen Anfangsbuchstaben haben muss, wird für solche Wörter einfach der Anfangsbuchstabe des Ursprungswortes benutzt. Wenn sich in dieser Liste kein Wort befindet, gibt es in der Wörterliste kein Wort, das dem „getwisteten“ Wort entspricht. Dann wird das „getwistete“ Wort, mit einem „#“ zur Erkennung vorangestellt, in den Text eingefügt. Das „#“ sorgt dafür, dass man weiß, dass dieses Wort nicht „enttwistet“ werden konnte. Befindet sich in der Liste genau ein Wort, so wird dieses genau so in den Text eingefügt, das getwistete Wort konnte erfolgreich dekodiert werden. Falls sich mehrere Wörter in der Liste befinden, werden diese, durch einen „/ “ getrennt, in den Text eingefügt. Dann muss der Leser entscheiden, welches Wort an dieser Stelle mehr Sinn macht.

2 Umsetzung

Das Sortieren der Wörterliste läuft folgendermaßen ab: Ein leeres dictionary wird erstellt. Mit einem for-loop wird über alle Wörter der Wörterliste iteriert. Dabei wird zunächst die Länge sowie der Anfangs- und Endbuchstabe bestimmt. Wenn die Länge größer als drei ist, wird per try-except versucht, mit der Länge als key auf die Wörterliste zuzugreifen. Wenn es bereits ein Wort derselben Länge gab, so funktioniert dies. Andernfalls wird mit der Länge als key ein leeres dictionary in unserem dictionary erzeugt. Im dictionary, auf das man über den key der Länge zugreifen kann, wird nun versucht, ein dictionary mit dem Anfangsbuchstaben als key zu betreten. Sollte dieses noch nicht existieren, wird ein Neues erstellt. In dem dictionary, das man über den Anfangsbuchstaben als key betritt, wird nun analog versucht, eine Liste mit dem Endbuchstaben als key zu betreten und im Falle ihrer Nichtexistenz erstellt. Schlussendlich wird das zu sortierende Wort in dieser Liste abgelegt. Nach Ende des Sortierens kann man auf eine Liste mit Wörtern der Länge n, Anfangsbuchstaben "a" und Endbuchstaben "z" also folgendermaßen zugreifen: `sorteddict[n]["a"]["z"]` Alle Wörter mit einer Länge kleiner als drei werden nicht in sorteddict übernommen, da diese Wörter vom Twisten/Enttwisten überhaupt nicht betroffen sind.

Um den Text zu twisten oder zu enttwisten, geht man nahezu exakt gleich vor. Der einzige Unterschied besteht darin, was mit den Wörtern mit mehr als vier Buchstaben geschieht: Dies wird in die Funktionen twistword und untwistword ausgelagert. Die Funktion twisttext benötigt also als erstes Argument den zu twistenden/enttwistenden Text und als zweites einen Boolean, ob der Text getwistet oder enttwistet werden soll. Zunächst wird ein leerer String definiert, zu dem der getwistete/enttwistete Text Stück für Stück hinzugefügt wird. Dann wird noch ein leerer String definiert, der immer bis zu einem vollständigen Wort ergänzt werden soll. Mit einem for-loop wird über jedes Zeichen im gesamten Text iteriert. Es wird überprüft, ob das Zeichen zu einem Wort gehört, indem mit der Python-Bibliothek für Reguläre Ausdrücke ermittelt wird, ob es sich um einen der 26 Buchstaben des Alphabets, um ein ä, ö, ü oder ein ß handelt. Dann wird dieser Buchstabe zum zweiten anfangs definierten String hinzugefügt. Sobald ein Zeichen gefunden wird, dass nicht zu einem Wort gehört, wird getestet, ob das Wort mehr als drei Buchstaben hat. Dann wird, in Abhängigkeit vom Eingangsboolean, das Wort der twistword-/untwistword-Funktion übergeben und das Ergebnis an unseren ersten String angehängt. Wenn das Wort weniger als vier Buchstaben hat, wird es einfach sofort an den ersten String angehängt. Danach wird der zweite String zurückgesetzt, da dieses Wort zu Ende ist. Außerdem wird das Zeichen, aus dem wir folgerten, dass das Wort zu Ende sein müsse, an den ersten String angehängt. Nach diesem Schema wird der ganze Text getwisted oder enttwisted.

Als nächstes betrachten wir die twistword-Funktion: Sie erstellt eine Liste aus den mittleren Buchstaben eines Wortes und mischt sie mithilfe des random-Moduls zufällig durch. Danach iteriert sie über die gemischte Liste und fügt die Buchstaben zum ersten Buchstaben hinzu. Schließlich fügt sie den letzten Buchstaben hinzu und gibt das getwistete Wort zurück.

Die untwistword-Funktion ist ein wenig komplizierter: Zunächst wird mithilfe der sortierten Wörterliste auf die Liste aller Wörter zugegriffen, welche die gleiche Länge sowie den gleichen Anfangs- und Endbuchstaben haben. Bei Großbuchstaben wird, wie oben erklärt, auch auf die Liste mit großem Anfangsbuchstaben zugegriffen. Das Zugreifen funktioniert über try-except-Konstrukte. Wenn das Zugreifen auf beispielsweise eine bestimmte Länge nicht funktioniert, weiß man, dass es keine Wörter mit dieser Länge in der sortierten Wörterliste gibt.

Hat man schließlich eine Liste mit in Frage kommenden Wörtern erstellt, so wird über die Wörter dieser Liste iteriert und überprüft, ob die mittleren Buchstaben eines Wortes ein Anagramm zu den mittleren Buchstaben des zu enttwistenden Wortes bilden. Dies geschieht mithilfe der Funktion middleanagramm.

Diese Funktion erhält als Input zwei zu vergleichende Wörter. Es wird mit einem for-loop über alle mittleren Buchstaben des einen Wortes iteriert und der aktuelle Buchstabe wird aus der Liste der mittleren Buchstaben des anderen Wortes herausgestrichen. Wenn in einer der beiden Listen Buchstaben übrigbleiben, sind die beiden Wörter keine Anagramme und die Funktion gibt False zurück, ansonsten True.

Danach wird über die Liste der noch übrigbleibenden Wörter iteriert. Es wird gezählt, wie oft ein Wort in der Liste vorkommt und dann so oft entfernt bis von jedes Wort noch genau einmal übrigbleibt. Schließlich wird die Anzahl der Wörter in der Liste gezählt. Die Ausgabe der Funktion wurde bereits im Lösungsansatz beschrieben.

3 Beispiele

3.1 Beispiel

Original: Der Twist (Englisch twist = Drehung, Verdrehung) war ein Modetanz im 4/4-Takt, der in den frühen 1960er Jahren populär wurde und zu Rock'n'Roll, Rhythm and Blues oder spezieller Twist-Musik getanzt wird.
Getwistet: Der Tiswt (Ecgsnlih twist = Dhrueng, Vernedruhg) war ein Mnoedtaz im 4/4-Tkat, der in den füerhn 1960er Jraehn ppäoulr wurde und zu Rock'n'Rlol, Rhyhtm and Buels oedr szepeellir Tiswt-Msuik gtaenzt wird.
Enttwisted: Der #Tiswt (Englisch #twist = Drehung, Verdrehung) war ein #Mnoedtaz im 4/4-Takt, der in den frühen/führen 1960er Jahren populär wurde und zu Rock'n'#Rlol, #Rhyhtm and Blues oder spezieller #Tiswt-Musik getanzt wird.
Kommentar: Man sieht, dass Fremdwörter im Allgemeinem nicht bekannt sind und dementsprechend die Ausgabe dem getwisteten Wort mit einem „#“ entspricht.

3.2 Beispiel

Original: Hat der alte Hexenmeister sich doch einmal wegbegeben! Und nun sollen seine Geister auch nach meinem Willen leben. Seine Wort und Werke merkt ich und den Brauch, und mit Geistesstärke tu ich Wunder auch.
Getwistet: Hat der atle Hetemsiexenr scih dcoh einaml wbgbeegeen! Und nun sellon sniee Gitseer acuh ncah mneeim Wllein leben. Sneie Wort und Wreke mkret ich und den Bacruh, und mit Gttseieksärse tu ich Wneudr auch.
Enttwisted: Hat der alte Hexenmeister sich doch einmal #wbgbeegeen! Und nun sollen seine Geister auch nach meinem Willen leben. Seine Wort und Werke merkt ich und den Brauch, und mit Geistesstärke tu ich Wunder auch.
Kommentar: Das Wort wegbegeben ist der Wörterliste unbekannt

3.3 Beispiel

Original: Ein Restaurant, welches a la carte arbeitet, bietet sein Angebot ohne eine vorher festgelegte Menüreihenfolge an. Dadurch haben die Gäste zwar mehr Spielraum bei der Wahl ihrer Speisen, für das Restaurant entstehen jedoch zusätzlicher Aufwand, da weniger Planungssicherheit vorhanden ist.
Getwistet: Ein Rnasauertt, whelces a la ctrae aeeirbtt, bietet sein Aeogbnt ohne enie vheorr feteggestle Mnogrleeüeifhne an. Dcudrah haebn die Gsäte zawr mher Silraeupm bei der Wahl irher Seepisn, für das Rnetuasrat etetshenn jdoech zstileczäuhr Aunfwad, da wegienr Puhnsgsareielnihct vodrnhaen ist.
Enttwistet: Ein Restaurant, welches a la #ctrae arbeitet/abrietet, bietet sein Angebot ohne eine vorher festgelegte #Mnogrleeüeifhne an. Dadurch haben die Gäste zwar mehr Spielraum bei der Wahl ihrer Speisen, für das Restaurant entstehen jedoch zusätzlicher Aufwand, da weniger #Puhnsgsareielnihct vorhanden ist.
Kommentar: Längere Zusammengesetzte Wörter sind in der Wörterliste ebenfalls nicht vorhanden

3.4 Beispiel

Original: Augusta Ada Byron King, Countess of Lovelace, war eine britische Adelige und Mathematikerin, die als die erste Programmiererin überhaupt gilt. Bereits 100 Jahre vor dem Aufkommen der ersten Programmiersprachen ersann sie eine Rechen-Mechanik, der einige Konzepte moderner Programmiersprachen vorwegnahm.
Getwistet: Astugua Ada Boyrn Knig, Constues of Laecolve, war eine bchisrtie Adligee und Mtamrkitaieehn, die als die esrte Pegarrimoriemrn üpbruaeht gilt. Betries 100 Jahre vor dem Afkmmoeun der eerstn Pgeparamcsmrerrihon enrsan sie eine Reehcn-Mahicnek, der eginie Ktzpnoee mrdeoenr Peersapcohamrmigrrn veoawgnhrm.
Enttwistet: #Astugua Ada #Boyrn #Knig, #Constues of #Laecolve, war eine britische Adelige und Mathematikerin, die als die erste Programmiererin überhaupt gilt. Bieters/Bereits/Breites 100 Jahre vor dem Aufkommen der ersten Programmiersprachen ersann sie eine Rechen-Mechanik, der einige Konzepte moderner Programmiersprachen vorwegnahm.

Kommentar: Man sieht, dass vor allem fremdsprachige Eigennamen unbekannt sind. Außerdem gibt es hier ein sehr schönes Beispiel für ein nicht eindeutiges Wort: „Bereits“ kann genauso auch als „Bieters“ oder „Breites“ dekodiert werden.

3.5 Beispiel

Der Auszug aus Alice im Wunderland wird hier nicht wiedergegeben, zusammenfassend lässt sich feststellen, dass Wörter der altdeutschen Rechtschreibung größtenteils nicht bekannt sind.

4 Quellcode

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
@author: Paddy Jo
@title BwInf 2018 A2
"""
#Für die Übergabe von Kommandozeilenargumenten
import sys
#Für Reguläre Ausdrücke
import re
#Zur Durchmischung der Buchstaben beim Twisten
import random
#Für das Handling von Unicode-Zeichen
import codecs

if len(sys.argv)!=2:
    print("Benutzung: Aufgabe2.py <pfad\\zur\\datei.txt>")
    sys.exit(0)
#Einlesen des Textes
with codecs.open(sys.argv[1],encoding="utf-8") as f:
    twisttext1=u""
    lines=f.readlines()
    for line in lines:
        if "\n" in line:
            line=line[:-1]+" "
        twisttext1+=line
#Einlesen der Wörterliste
with codecs.open("Aufgabe2\\woerterliste.txt","r",encoding="utf-8") as f:
    woerterliste=[line.rstrip('\n') for line in f]

#Sortieren der Wörterliste
sorteddict={}
for word in woerterliste:
    len1 = len(word)
    if len1 > 3:
        firstchar = word[0]
        lastchar = word[-1]
        try:
            a = sorteddict[len1]
            try:
                a = sorteddict[len1][firstchar]
                try:
                    a = sorteddict[len1][firstchar][lastchar]
                except:
                    sorteddict[len1][firstchar][lastchar] = []
            except:
                sorteddict[len1][firstchar] = {}
                sorteddict[len1][firstchar][lastchar] = []
        except:
            sorteddict[len1] = {}
            sorteddict[len1][firstchar] = {}
            sorteddict[len1][firstchar][lastchar] = []

        sorteddict[len1][firstchar][lastchar].append(word)


def middleanagramm(word1,word2):
    """
    Input:  Zwei Wörter
```

```
60      Output: True  wenn die mittleren Buchstaben der beiden Wörter ein Anagramm bilden
                False ansonsten
62      """
        wordlist1=list(word1)[1:-1]
64      wordlist2=list(word2)[1:-1]
        for i in wordlist1:
66          if i in wordlist2:
                wordlist2.remove(i)
68          else:
                return False
70      if len(wordlist2)>0:
            return False
72      return True

74
    def twistword(word):
76      """
        Input:  ein Wort
78      Output: ein Wort, bei dem alle mittleren Buchstaben des Eingabewortes
                zufällig permutiert sind
80      """
        tlist=list(word[1:-1])
82      random.shuffle(tlist)
        wordend=word[-1]
84      word=word[0]
        for i in tlist:
86          word+=i
        word+=wordend
88      return word

90
    def untwistword(tword):
92      """
        Input:  Ein getwistetes Wort (Die mittleren Buchstaben sind zufällig permutiert)
94      Output: Das Eingabewort mit einem "#" vorangestellt,
                wenn in der Wörterliste kein Wort zu finden ist,
96              dessen Länge, Anfangs- und Endbuchstabe gleich sind und dessen
                mittlere Buchstaben ein Anagramm zum Eingabewort bilden
98              (oder, bei großgeschriebenen Wörtern auch Wörter, bei denen der
                Anfangsbuchstabe in der Wörterliste kleingeschrieben wird)
100
                Das enttwistete Eingabewort, wenn es genau ein Wort gibt,
102             das die genannten Kriterien erfüllt

104             Mehrere Wörter, die mit einem "/" getrennt werden,
                wenn es mehrere Wörter gibt, die die genannten Kriterien erfüllen
106
        """
108     possiblewords2 = []

110     if tword[0].isupper():
            upper = True
112         #gibt es wörter mit großem anfangsbuchstaben?
            try:
114             possiblewords1 = sorteddict[len(tword)][tword[0]][tword[-1]]
            except:
116             #es gibt keine wörter mit großem anfangsbuchstaben
                upper = False
118             try:
                    #gibt es wörter mit kleinem anfangsbuchstaben?
120                 possiblewords1 = sorteddict[len(tword)][tword[0].lower()][tword[-1]]
                except:
122                 #keins von den wörtern passt
                    return "#"+tword
124         #es gibt wörter mit großem Anfangsbuchstaben
            if upper:
126             try:
                    possiblewords1 += sorteddict[len(tword)][tword[0].lower()][tword[-1]]
128             except:
                    pass
130     else:

132         try:
```

```
            possiblewords1 = sorteddict[len(tword)][tword[0]][tword[-1]]
        except:
            return "#"+tword

    for word in possiblewords1:
        if middleanagramm(tword,word):
            possiblewords2.append(tword[0]+word[1:])

    #mehrfach vorkommende wörter aus einer liste entfernen
    for i in possiblewords2:
        n = possiblewords2.count(i)
        for x in range(n-1):
            possiblewords2.remove(i)

    if len(possiblewords2) == 0:
        return "#"+tword

    if len(possiblewords2) > 1:
        string = u""
        for n,i in enumerate(possiblewords2):
            if n != 0:
                string += "/"
            string += i
        return string

    else:
        return possiblewords2[0]


def twisttext(twtext,twist):
    """
    iteriert über alle Buchstaben des Textes, gruppiert Wörter und
    twisted/enttwisted (in Abhängigkeit der Variable twist)
    alle Wörter mit über drei Buchstaben.
    """

    retstring = u""
    word = ""
    for i in twtext:
        if re.search("[A-Za-zÄäÖöÜüß]",i):
            word += i
        else:
            if len(word)>3:
                if twist:
                    retstring += twistword(word)
                else:
                    retstring += untwistword(word)
            else:
                retstring += word
            word = u""
            retstring += i
    return retstring

print(twisttext1)
a = twisttext(twisttext1,True)
print(a)
a = twisttext(a,False)
print(a)
```

Aufgabe2.py